Name: Braydyn Proctor Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

[**Instructions**: Remove everything that is not a heading below and fill in with your own diagrams, etc.]

Braydyn - Proc generation, Level Designer

## Brief introduction \_\_/3

For my feature for the game InBetween I will be working on the pseudo-procedural level generation that stitches together pre-designed rooms in different patterns to form a dynamic level.

Players will spawn in a randomly selected room from an array Of spawn rooms. The spawn room will have various points where other random objects can generate in. The room will have 2-3 doors. These doors will lead to other rooms after they have been interacted with. (i.e opening the door)

The rooms that will generate off the door opening will have different factors pertaining to what type of room it is. (boss room, loot room, safe room, shop (etc.)). Each type will have at least 4 base layouts, i.e. the setup of walls. Everything in the room will be generated off these types of rooms.

In addition to this system I am also responsible for designing all of these rooms and their base layouts.

## Use case diagram with scenario \_\_14

[Use the lecture notes in class.

Ensure you have at least one exception case, and that the <<extend>> matches up with the Exceptions in your scenario, and the Exception step matches your Basic Sequence step.

Also include an <<include>> that is a suitable candidate for dynamic binding]

Example:

### Use Case Diagrams

A diagram of a company

AI-generated content may be incorrect.

### Scenarios

**[You will need a scenario for each use case]**

**Name:** Level Generation

**Summary:** The Player will Open a Door and Generate a new room.

**Preconditions:** The Player has Spawned into the spawn room

**Basic sequence:**

**Step 1:** Player Is Spawned in and walks toward the door

**Step 2:** Player interacts with door

**Step 3:** Randomly Choose Room Type.

**Step 4:** Detect where spawn points should be in relation to the layout of the room

**Step 5:** Determine How many doors and where they should go

**Step 6:** Randomly choose if each point is a empty tile, furniture tile, or enemy spawn point.

**Step 7:** Instantiate All Objects & Spawn Points

**Exceptions:**

**Step 1:** Error Generating Objects (Use Fallback Objects)

**Step 2:** Room Chosen is out of scope, (Use room 0)

**Post conditions:** Calculated value is displayed.

**Priority:** 2\*

**ID:** C02

\*The priorities are 1 = must have, 2 = essential, 3 = nice to have.

## Data Flow diagram(s) from Level 0 to process description for your feature \_\_\_\_\_\_\_14

[Get the Level 0 from your team. Highlight the path to your feature]

Example:

### Data Flow Diagrams

A diagram of a game

AI-generated content may be incorrect.

### 

### Process Descriptions

PROCESS DoorWaitForInput()

IF object.tag == "Player" AND KeyPressed("E") THEN

CALL GenerateRoom()

ELSE

// Do nothing

END IF

END PROCESS

PROCESS GenerateRoom()

roomType ← RANDOM(1,5) // Select a random room type

CALL SetRoomType(roomType)

IF roomType IS INVALID THEN

roomType ← 1 // Fallback

CALL LogError("Invalid room type selected, using default Room 1")

END IF

success ← CALL PlaceObjectsInRoom() // Populate the room

IF success == FALSE THEN

CALL ApplyFallbackObjects()

CALL LogError("Error in object placement, using fallback objects")

END IF

success ← CALL FinalizeRoom() // Ensure room is connected properly

IF success == FALSE THEN

CALL ReloadPreviousRoom()

CALL LogError("Room finalization failed, reloading previous room state")

END IF

END PROCESS

## Acceptance Tests \_\_\_\_\_\_\_\_9

[Describe the inputs and outputs of the tests you will run. Ensure you cover all the boundary cases.]

Test1: Room Type Selection

* Run the selection algorithm 100 times to test fail percentage rate.
* Expected Output:
  + Every room time should appear at least 10 times.
  + No room should appear more than 30 times.
  + Room 1 should shoot a debug error when selected because of an error.

Test 2: Object & Spawnpoint Placement:

* Generate 50 room layouts for each type of room
* Expected Output:
  + 90% of the layouts should be valid and have enemies and objects.
  + No more than 5% of rooms should have to rely on fallback obj

Test 3: Room Connections

* Generate 200 rooms all interconnected
* Expected Output:
  + Each room has at least 1 door.
  + Shouldn’t be any dead ends unless it’s a boss room or treasure room
  + All rooms should connect to another room.

Test 4: Error Handling Tests:

* Intentionally Break systems by overloading / giving it bad information
* Expected Outcome:
  + Error should log with the values used.
  + Error should force game to use fallback room.
  + Game should continue as normal.

## Timeline \_\_\_\_\_\_\_\_\_/10

[Figure out the tasks required to complete your feature]

Example:

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (hrs) | Predecessor Task(s) |
| 1. Requirements Collection | 4 | - |
| 2. Room Design & Layout Creation | 6 | 1 |
| 3. Object & Enemy spawn point Logic | 8 | 1 |
| 4. Procedural-Stitching Generation Programming | 14 | 2,3 |
| 5. Error Handlers Implementation | 4 | 4 |
| 6. Testing Debugging | 5 | 4,5 |
| 7. Optimization | 7 | 6 |
| 8. Documentation | 3 | 7 |

### Pert diagram

A diagram of a diagram

AI-generated content may be incorrect.

### Gantt timeline

A graph with blue rectangles

AI-generated content may be incorrect.